

Algorithm Design and Analysis

**CSE
565**

LECTURE 17

Dynamic Programming

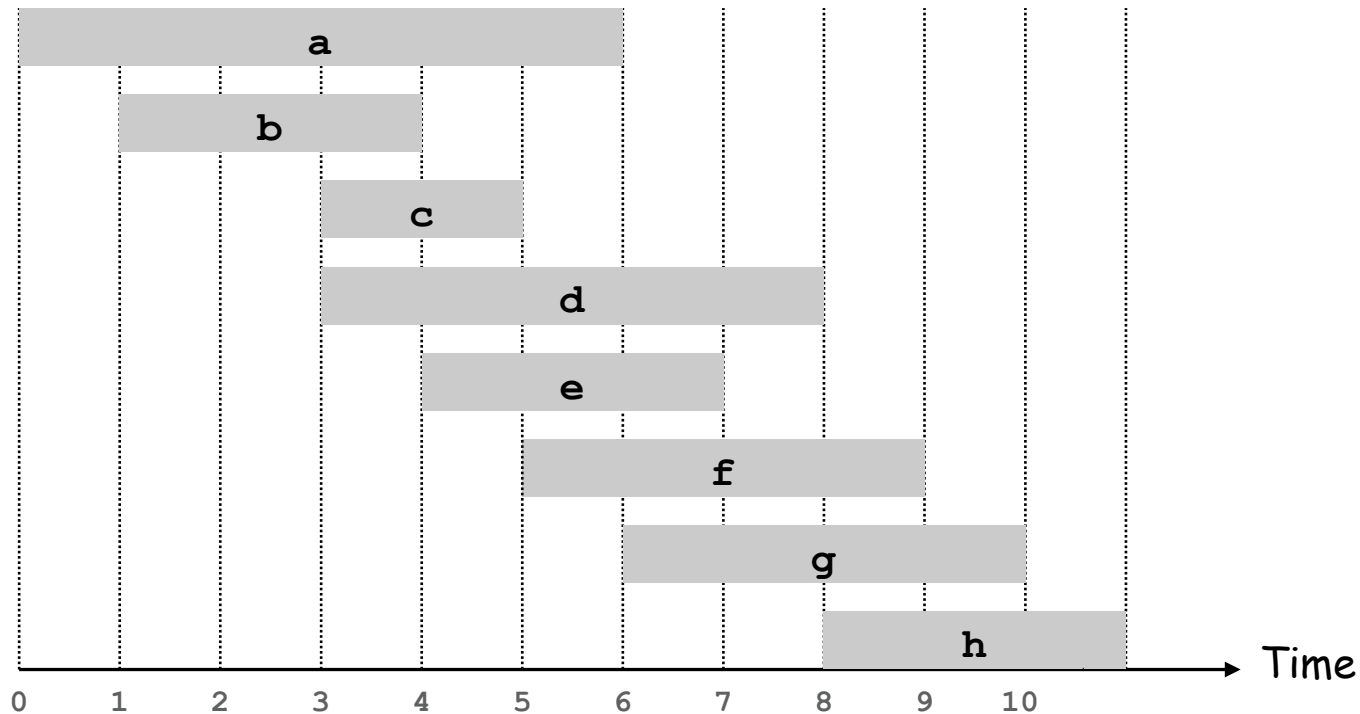
- (WIS recap)
- Segmented Least Squares

Adam Smith

Weighted Interval Scheduling

Weighted interval scheduling problem.

- Job j starts at s_j , finishes at f_j , and has weight or value v_j .
- Two jobs **compatible** if they don't overlap.
- Goal: find maximum **weight** subset of mutually compatible jobs.



$$\text{OPT}(j) = \max(v_i + \text{OPT}(p(j)) , \text{OPT}(j-1))$$

Weighted Interval Scheduling: Memoization

Memoization. Store results of each sub-problem in a table; lookup as needed.

```
Input:  $n, s_1, \dots, s_n, f_1, \dots, f_n, v_1, \dots, v_n$ 
```

```
Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .
```

```
Compute  $p(1), p(2), \dots, p(n)$ 
```

```
for  $j = 1$  to  $n$ 
```

```
     $M[j] = \text{empty}$ 
```

```
 $M[0] = 0$ 
```

← global array

```
M-Compute-Opt( $j$ ) {
```

```
    if ( $M[j]$  is empty)
```

```
         $M[j] = \max(w_j + \text{M-Compute-Opt}(p(j)), \text{M-Compute-Opt}(j-1))$ 
```

```
    return  $M[j]$ 
```

```
}
```

Equivalent algorithm: Bottom-Up

Bottom-up dynamic programming. Unwind recursion.

```
Input:  $n, s_1, \dots, s_n, f_1, \dots, f_n, v_1, \dots, v_n$ 
```

```
Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .
```

```
Compute  $p(1), p(2), \dots, p(n)$  ← how fast?
```

```
Iterative-Compute-Opt {  
    M[0] = 0  
    for j = 1 to n  
        M[j] = max( $v_j + M[p(j)]$ , M[j-1])  
}
```

Total time = (sorting) + $O(n)$ = $O(n \log(n))$

Weighted Interval Scheduling: Finding a Solution

Q. Dynamic programming algorithms computes optimal value.

What if we want the solution itself?

A. Do some post-processing.

```
Run M-Compute-Opt(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if ( $v_j + M[p(j)] > M[j-1]$ )
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

- # of recursive calls $\leq n \Rightarrow O(n)$.

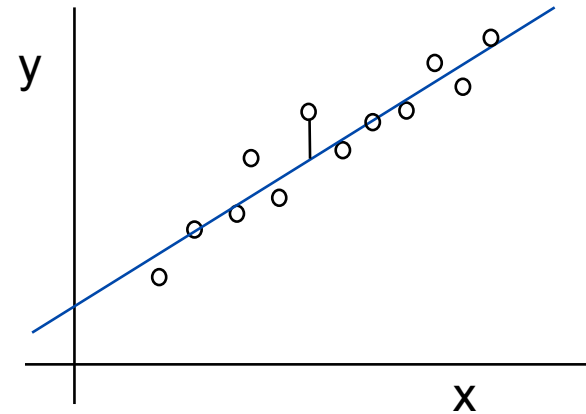
6.3 Segmented Least Squares

Segmented Least Squares

Least squares.

- Foundational problem in statistic and numerical analysis.
- Given n points in the plane: $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.
- Find a line $y = ax + b$ that minimizes the sum of the squared error:

$$SSE = \sum_{i=1}^n (y_i - ax_i - b)^2$$



Solution. Calculus \Rightarrow min error is achieved when

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i) (\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n} \leftarrow O(n) \text{ time}$$

Segmented Least Squares

Segmented least squares.

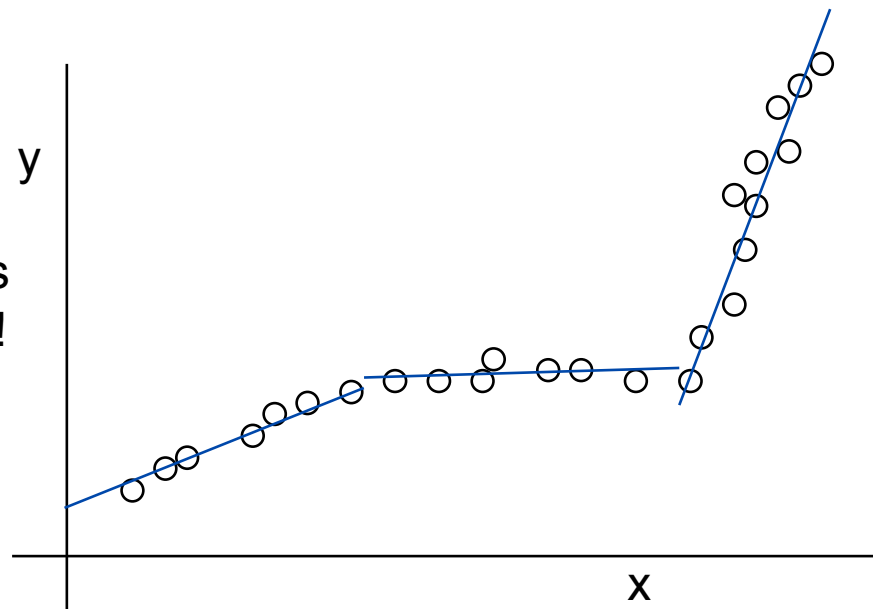
- Points lie roughly on a sequence of several line segments.
- Given n points in the plane $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ with
- $x_1 < x_2 < \dots < x_n$, find a sequence of lines that minimizes $f(x)$.

Q. What's a reasonable choice for $f(x)$ to balance accuracy and parsimony?

↑
number of lines

↑
goodness of fit

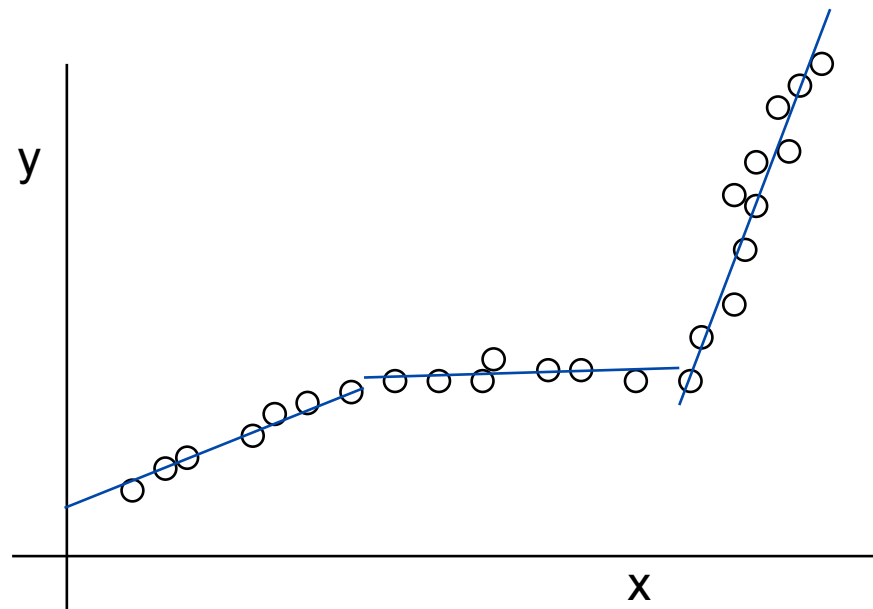
Note: discontinuous functions permitted!



Segmented Least Squares

Segmented least squares.

- Points lie roughly on a sequence of several line segments.
- Given n points in the plane $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ with
- $x_1 < x_2 < \dots < x_n$, find a sequence of lines that minimizes:
 - the sum of the sums of the squared errors E in each segment
 - the number of lines L
- Tradeoff function: $E + cL$, for some constant $c > 0$.



Dynamic Programming: Multiway Choice

Notation.

- $OPT(j)$ = minimum cost for points p_1, p_{i+1}, \dots, p_j .
- $e(i, j)$ = minimum sum of squares for points p_i, p_{i+1}, \dots, p_j .

To compute $OPT(j)$:

- Last segment uses points p_i, p_{i+1}, \dots, p_j for some i .
- Cost = $e(i, j) + c + OPT(i-1)$.

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min_{1 \leq i \leq j} \{ e(i, j) + c + OPT(i-1) \} & \text{otherwise} \end{cases}$$

Segmented Least Squares: Algorithm

```
INPUT:  $n, p_1, \dots, p_N, c$ 

Segmented-Least-Squares() {
  M[0] = 0
  for j = 1 to n
    for i = 1 to j
      compute the least square error  $e_{ij}$  for
      the segment  $p_i, \dots, p_j$ 

  for j = 1 to n
    M[j] =  $\min_{1 \leq i \leq j} (e_{ij} + c + M[i-1])$ 

  return M[n]
}
```

Running time. $O(n^3)$. ← can be improved to $O(n^2)$ by pre-computing $e_{i,j}$'s

- Bottleneck = computing $e(i, j)$ for $O(n^2)$ pairs, $O(n)$ per pair using previous formula.